Ontology-based and Evolutionary Search for Computational Agents Schemes

Roman Neruda*

Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodarenskou vezi 2, 18207 Prague 8, Czech Republic
roman@cs.cas.cz

Abstract

This work deals with a problem of automatic composition of multi-agent system satisfying given constraints. A general concept of representation of connected groups of agents (schemes) within a multi-agent system is introduced and utilized for automatic building of schemes to solve a given computational intelligence task. We propose a combination of an evolutionary algorithm and a formal logic resolution system which is able to propose and verify new schemes. The approach is illustrated on simple examples.

1 Introduction

Autonomous agents are small self-contained programs that can solve simple problems in a well-defined domain [10]. In order to solve complex problems, agents have to collaborate, forming Multi-Agent Systems (MAS). A key issue in MAS research is how to generate MAS configurations that solve a given problem [5]. In most Systems, an intelligent (human) user is required to set up the system configuration. Developing algorithms for automatic configuration of Multi-Agent Systems is a major challenge for AI research.

We have developed a platform for creating Multi-Agent Systems [7], [9]. Its main areas of application are computational intelligence methods (genetic algorithms, neural networks, fuzzy controllers) on single machines and clusters of workstations. Hybrid models, including combinations of artificial intelligence methods such as neural networks, genetic algorithms and fuzzy logic controllers, seem to be a promising and extensively studied research area [2]. Our distributed multi-agent system — provides a support for an easy creation and execution of such hybrid AI models utilizing the Java/JADE environment.

The above mentioned applications require a number of cooperating agents to fulfill a given task. So far, MAS are created and configured manually. In this paper, we introduce two approaches for creation and possible configuration of MAS. One of them is based on formal descriptions and provides a logical reasoning component for the system.

The second approach to MAS generation employs evolutionary algorithm (EA) which is tailored to work on special structures—directed acyclic graphs—denoting MAS schemata. The advantage of EA is that it requires very little additional information apart from a measure of MAS performance. Thus, the typical run of EA consists of thousands of simulations which build and assess the fitness values of various MAS. Since the properties of logical reasoning search and evolutionary search are dual, the ultimate goal of this work is to provide a solution combining these two approaches in a hybrid search algorithm. This paper presents the first steps towards such a goal.

2 Description of MAS by means of Logics

The most natural approach to formalize ontologies is the use of First Order Predicate Logics (FOL). The disadvantage of FOL-based languages is the expressive power of FOL. FOL is undecidable [4], and there are no efficient reasoning procedures. Nowadays, the de facto standard for ontology description language for formal reasoning is the family of description logics. Description logics are equivalent to subsets of first order logic restricted to predicates of arity one and two [3]. They are known to be equivalent to modal logics [1]. For the purpose of describing multi-agent systems, description logics are sometimes too weak. In these cases, we want to have a more expressive formalism. We decided to use Prolog-style logic programs for this. In the following we describe how both approaches can be combined together.

An *agent* is an entity that has some form of perception of its environment, can act, and can communicate with other agents. It has specific skills and tries to achieve goals. A *Multi-Agent System (MAS)* is an assemble of interacting agents in a common environment [6]. In order to use automatic reasoning on a MAS, the MAS must be described in formal logics. For the computational system, we define a formal description for the static characteristics of the agents, and their communication

*This research has been supported by the the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realization".

channels. We do not model dynamic aspects of the system yet.

Agents communicate via messages and triggers. Messages are XML-encoded FIPA standard messages. Triggers are patterns with an associated behavior. When an agent receives a message matching the pattern of one of its triggers, the associated behavior is executed. In order to identify the receiver of a message, the sending agent needs the message itself and an id of the receiving agent. A conversation between two agents usually consists of a number of messages conforming to FIPA protocols. In order to abstract from the actual messages, we subsume all these messages under a *message type* when describing an agent in formal logics.

Definition 1 (Message type) *A* message type *identifies a category of messages that can be send to an agent in order to fulfill a specific task. We refer to message types by unique identifiers.*

The set of message types understood by an agent is called its *interface*. For outgoing messages, each link of an agent is associated with a message type. Via this link, only messages of the given type are sent. We call a link with its associated message type a *gate*.

Definition 2 (Interface) *An interface is the set of message types understood by a class of agents.*

Definition 3 (Gate) *A gate is a tuple consisting of a message type and a named link.*

Now it is easy to define if two agents can be connected: Agent A can be connected to agent B via gate G if the message type of G is in the list of interfaces of agent B. Note that one output gate sends messages of one type only, whereas one agent can receive different types of messages. This is a very natural concept: When an agent sends a message to some other agent via a gate, it assigns a specific role to the other agent, e.g. being a supplier of training data. On the receiving side, the receiving agent usually should understand a number of different types of messages, because it may have different roles for different agents.

Definition 4 (Connection) *A connection is described by a triple consisting of a sending agent, the sending agent's gate, and a receiving agent.*

Next we define *agents* and *agent classes*. Agents are created by generating instances of classes. An agent derives all its characteristics from its class definition. Additionally, an agent has a name to identify it. The static aspects of an agent class are described by the interface of the agent class (the messages understood by the agents of this class), the gates of the agent (the messages send by agents of this class), and the type(s) of the agent class. Types are nominal identifiers for characteristics of an agent. The types used to describe the characteristics of the agents should be ontological sound.

Concepts	
mas(C)	C is a Multi-Agent System
class(C)	C is the name of an agent class
gate(C)	C is a gate
m_type(C)	C is a message type
Roles	
type(X,Y)	Class X is of type Y
has_gate(X,Y)	Class X has gate Y
gate_type(X,Y)	Gate X accepts messages of type Y
interface(X,Y)	Class X understands mess. of type Y
instance(X,Y)	Agent X is an instance of class Y
has_agent(X,Y)	Agent Y is part of MAS X

Table 1. Concepts and roles used to describe MAS.

```
class(decision_tree)
type(decision_tree, computational_agent)
has_gate(decision_tree, data_in)
gate_type(data_in, training_data)
interface(decision_tree, control_messages)
```

Figure 1. Example agent class definition.

Definition 5 (Agent Class) *An agent class is defined by an interface, a set of message types, a set of gates, and a set of types.*

Definition 6 (Agent) *An agent is an instance of an agent class. It is defined by its name and its class.*

A Multi-Agent System can be described by three elements: The set of agents in the MAS, the connections between these agents, and the characteristics of the MAS. The characteristics (constraints) of the MAS are the starting point of logical reasoning: In *MAS checking* the logical reasoner deduces if the MAS fulfills the constraints. In *MAS generation*, it creates a MAS that fulfills the constraints, starting with an empty MAS, or a manually constructed partial MAS.

Definition 7 (Multi-Agent System) Multi-Agent Systems (MAS) *consist of a set of agents, a set of connections between the agents, and the characteristics of the MAS.*

In order to describe agents and Multi-Agent Systems in description logics, the definitions 1 to 7 are mapped onto description logic concepts and roles as shown in table 1. An example agent class description is given in figure 1. It defines the agent class "decision_tree". This agent class accepts messages of type "control_message". It has one gate called "data_in" for data agent and emits messages of type "training_data".

In the same way, A-Box instances of agent classes are defined: $instance(decision\_tree, dt\_instance)$ An agent is assigned to a MAS via role "has_agent". In the following example, we define "dt_instance" as belonging to MAS "my_mas": $has\_agent(my\_mas, dt\_instance)$

Since connections are relations between three elements, a sending agent, a sending agent's gate, and a receiving agent,

we can not formulate this relationship in traditional description logics. It would be possible to circumvent the problem by splitting the triple into two relationships, but this would be counter-intuitive to our goal of defining MAS in an ontological sound way. Connections between agents are relationships of arity three: Two agents are combined via a gate. Therefore, we do not use description logics, but traditional logic programs in Prolog notation to define connections: $connection(dt\_instance, other\_agent, gate)$

Constraints on MAS can be described in Description Logics, in Prolog clauses, or in a combination of both. As an example, the following concept description requires the MAS "dt_MAS" to contain a decision tree agent: $dt\_MAS \sqsupseteq mas \sqcap has\_agent.(\exists instance.decision\_tree)$

An essential requirement for a MAS is that agents are connected in a sane way: An agent should only connect to agents that understand its messages. According to definition 4, a connection is possible if the message type of the sending agent's output gate matches a message type of the receiving agent s interface. With the logical concepts and descriptions given in this section, this constraint can be formulated as a Prolog style horn rule. If we are only interested in checking if a connection satisfies this property, the rule is very simple:

connection(S,R,G) ←
    instance(R, RC) ∧
    instance(S, SC) ∧
    interface(RC, MT)∧
    has_gate(SC, G) ∧
    gate_type(G, MT)

The following paragraphs show an example for logical descriptions of MAS. *Computational MAS:* A computational MAS can be defined as a MAS with a task manager, a computational agent and a data source agent which are interconnected.

comp_MAS(MAS) ←
    type(CAC, computational_agent)∧
    instance(CA, CAC)∧
    has_agent(MAS, CA)∧
    type(DSC, data_source)∧
    instance(DS, DSC)∧
    has_agent(MAS, DS)∧
    connection(CA, DS, G)∧
    type(TMC, task_manager)∧
    instance(TMC, TM)∧
    has_agent(MAS, TM)∧
    connection(TM, CA, GC)∧
    connection(TM, DS, GD)

3 Evolutionary search

The proposed evolutionary algorithm operates on schemes definitions in order to find a suitable scheme solving a specified problem. The genetic algorithm has three inputs: First, the number and the types of inputs and outputs of the scheme. Second, the *training set*, which is a set of prototypical inputs and the corresponding desired outputs, it is used to compute the fitness of a particular solution. And third, the list of types of building blocks available for being used in the scheme.

We supply three operators that would operate on graphs representing schemes: *random scheme creation*, *mutation* and *crossover*. The aim of the first one is to create a random scheme. This operator is used when creating the first (random) generation. The diversity of the schemes that are generated is the most important feature the generated schemes should have. The goal of the crossover operator is to create offsprings from two parents. The crossover operator proposed for scheme generation creates one offspring. The operator horizontally divides the mother and the father, takes the first part from father's scheme, and the second from mother's one. The mutation operator is very simple. It finds two links in the scheme (of the same type) and switches their destinations.

4 Experiments

This section describes the experiments we have performed with generating the schemes using the genetic algorithm described above.

The training sets used for experiments represented various polynomials. The genetic algorithm was generating the schemes containing the following agents representing arithmetical operations: *Plus* (performs the addition on floats), *Mul* (performs the multiplication on floats), *Copy* (copies the only input (float) to two float outputs), *Round* (rounds the incoming float to the integer) and finally *Floatize* (converts the int input to the float).

The selected set of operators has the following features: it allows to build any polynomial with integer coefficients. The presence of the *Round* allows also another functions to be assembled. These functions are the 'polynomials with steps' that are caused by using the *Round* during the computation.

The results of the experiments depended on the complexity of the desired functions. The functions, that the genetic algorithm learned well and quite quickly were functions like $x^3 - x$ or x^2y^2. The learning of these functions took from tens to hundred generations, and the result scheme precisely computed the desired function.

Also more complicated functions were successfully evolved. Having in mind, that the only constant that can be used in the scheme is -1, we can see, that the scheme is quite big (comparing to the previous example where there was only approximately 5–10 building blocks) — see Fig. 2. It took much more time/generations to achieve the maximal fitness, namely 3000 in this case.

On the other hand, learning of some functions remained in the local maxima, which was for example the case of the function $x^2 + y^2 + x$.

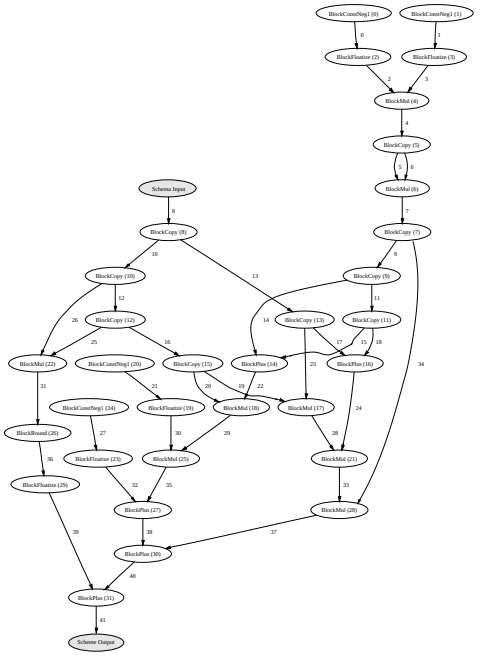

Figure 2. Function $x^3 - 2x^2 - 3$. The scheme with fitness 1000 (out of 1000), taken from 3000th generation.

5 Conclusions

We have presented a hybrid system that uses a combination of evolutionary algorithm and a resolution system to automatically create and evaluate multi-agent schemes. So far, the implementation has focused on relatively simple agents computing parts of arithmetical expressions. Nevertheless, the sketched experiments demonstrate the soundness of the approach. A similar problem is described and tackled in [11] by means of matchmaking in middle-agents where authors make use of ontological descriptions but utilize other search methods than EA.

In our future work we plan to extend the system in order to incorporate more complex agents into the schemes. Our ultimate goal is to be able to propose and test schemes containing a wide range of computational methods from neural networks to fuzzy controllers, to evolutionary algorithms. While the core of the proposed algorithm will remain the same, we envisage some modifications in the genetic operators based on our current experience.

Namely, a finer consideration of parameter values, or configurations, of basic agents during the evolutionary process needs to be addressed. So far, the evolutionary algorithm rather builds the -3 constant by combining three agents representing the constant 1, than modifying the constant agent to represent the -3 directly. We hope to improve this behavior by introducing another kind of genetic operator. This mutation-like operator can be more complicated in the case of real computational agents such as neural networks, though. Nevertheless, this approach can reduce the evolutionary algorithm search space substantially.

We also plan to extend the capabilities of the resolution system towards more complex relationship types than the ones described in this paper. In our work [8] we use ontologies for the description of agent capabilities, and have the CSP-solver reason about these ontologies. The next goal is to provide hybrid solution encompassing the evolutionary algorithm enhanced by ontological reasoning.

References

[1] F. Baader. Logic-based knowledge representation. In M. J. Wooldrige and M. Veloso, editors, *Artificial Intelligence Today, Recent Trends and Developments*, pages 13–41. Springer, 1999.

[2] P. Bonissone. Soft computing: the convergence of emerging reasoning technologies. *Soft Computing*, 1:6–18, 1997.

[3] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[4] M. Davis, editor. *The Undecidable—Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Raven Press, 1965.

[5] J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[6] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Harlow: Addison Wesley Longman, 1999.

[7] Pavel Krušina, Roman Neruda, and Zuzana Petrova. More autonomous hybrid models in bang. In *International Conference on Computational Science (2)*, pages 935–942, 2001.

[8] R. Neruda and G. Beuster. Towards dynamic generation of computational agents by means of logical descriptions. In *MASUPC'07 – International Workshop on Multi-Agent Systems Challenges for Ubiquitous and Pervasive Computing*, pages 17–28, 2007.

[9] Roman Neruda, Pavel Krušina, Petra Kudova, and Gerd Beuster. Bang 3: A computational multi-agent system. In *Proceedings of the 2004 WI-IAT'04 Conference*. IEEE Computer Society Press, 2004.

[10] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(2):205–244, 1995.

[11] Zili Zhang and Chengqi Zhang. *Agent-Based Hybrid Intelligent Systems*. Springer Verlag, 2004.